Homework 2

1) Why you should be interested in learning about Lambda calculus?

Ans)

→1) Lambda calculus can encode any computation. If you write a program in any programming language, which has ever been invented, or ever will be invented, or really any sequential programming language, it can in some way be encoded in the Lambda Calculus; and of course it may be

extremely inefficient when you do that, but that is not the point. This is a basic idea of computation, and we want to think how many and what kind of programs can we encode in this, and actually you can encode anything. This is kind of Church-turing hypothesis.

2) Lambda calculus can be regarded as the basis for functional programming languages like Haskell.

So these are becoming increasingly popular these days and actually a very sophisticated language like Haskell is compiled down to a very small core language, which is essentially a glorified form of Lambda calculus. So, if interested in functional programming languages like Haskell, or the ML family, these are all fundamentally based on Lambda calculus - it is just kind of a glorified syntax on top of that.

3) The Lambda calculus is actually now present in most major programming languages.

So, this was not the case 10 or 15 years ago, but is the case today. So, if we look at

languages like Java, C#, even Visual Basic, F# and so on, all of these languages now encode Lambda Calculus, or include Lambda calculus as a fundamental component. So every computer scientist today needs to learn about Lambda calculus.

Question

2) How do you encode the concepts of TRUE, FALSE, NOT, AND, OR ?

Ans) The Y combinator is a fixed-point combinator in lambda calculus that allows for the recursive definition of functions. However, it is not clear like how to encode logical operators using Y combinator, it is possible to define functions that describe TRUE, FALSE, NOT, AND, OR concepts.

TRUE and FALSE are the functions which take two arguments and then returns first and second argument respectively.

In Lambda calculus notation,

$$\boxed{\begin{aligned} TRUE &= \lambda x.\lambda y.x \\ FALSE &= \lambda x.\lambda y.y \end{aligned}}$$

Whereas, NOT function takes a boolean value and returns opposite

$$\boxed{NOT = \lambda b.b \ false \ true}$$

Here, it is taking a boolean value 'b' and then applies it to 'false' and 'true' functions in reverse order.

If 'b' is 'false', 'true' will be returned

If 'b' is true, 'false' will be returned.

For 'AND' and 'OR', we define them as functions which take two boolean values and then return the result of applying the respective logical operation:

$$\boxed{\begin{aligned} AND &= \lambda_{b1} . \lambda_{b2} . b1 \;\; b2 \;\; false \\ OR &= \lambda_{b1} . \lambda_{b2} . b1 \;\; true \;\; b2 \end{aligned}}$$

Here, 'AND' function is taking two boolean values 'b1' and 'b2', and then applies 'b1' to 'b2' and 'false'.
If 'b1' is true, result is 'b2'
or else it will be 'false'
'OR' function is taking two boolean values 'b1' and 'b2', and then applies 'true' to 'b1' and 'b2'.
If 'b1' is true, result is 'true'
or else it will be 'b2'

3) What is important about the Lambda calculus expression called 'Y combinator'?

Ans) Y combinator is known as important expression in Lambda calculus because it is a way of doing recursion in a language which does not have recursion. Functions are described as expressions in the Lambda calculus that takes one or more parameters and provide a result based on those inputs. However, it can be challenging to construct functions that refer to themselves because the language does not provide named variables or functions.

Y combinator is called as fixed-point combinator that provides a way to define recursive functions in the Lambda calculus. It accepts a function as an argument and outputs a new function with identical behavior but with the abilility to refer to itself. This makes it feasible to design more complex functions and computations by allowing the definition of functions that can be applied to themselves recursively.

4) Write the Ycombinator expression in Lambda calculus.

Ans) Y combinator expression in Lambda Calculus is:-

$$rec = \lambda f.(\lambda x.f(x\,x))\ (\lambda x.f(x\,x))$$

It is exactly the same idea of self application. The only difference is we have got 'f' in the way, which we are going to be repeatedly applying when we do the recursion.
So, this is the Y combinator. It is not recursive, but it encodes recursion. This is very simple but powerful idea and we can do this in most of the programming languages. This gives a way of doing recursion in a language which doesn't have any recursion at all.

5) Where did the language 'Haskell' get its name?

Ans) The programming language 'Haskell' got its name after Haskell Curry, a mathematician and logician who made substantial contributions to the Lambda calculus development. Haskell and many other functional programming languages, including those based on function abstraction and application, are theoretically based on Lambda calculus. Haskell Curry's contribution towards the Lambda Calculus helped in laying foundation for functional programming, and his name was selected in order to recognize his work in this field.

6) In the video it was mentioned that Erlang was used to code what?

Ans) Erlang was used to do 'Quickcheck', it lets to write test code that says what your program should do and then it generates as many tests to check that it does. Erlang worked on this for many years.

7) How is pattern matching used?

Pattern matching is a way of deciding which definition of a function to apply in a given case depending on the structure of the input

patterns

sum[n] = n

sum(n, ns) = n + sum ns

sum[4, 5, 2]

n

ns

= 4 + sum[5, 2]

= 4 + (5 + sum[2])

= 4 + (5 + 2)

= 4 + 7

= 11

Pattern matching is one of the techniques which is used to find and match a specific pattern within a larger piece of data. Pattern matching is used in many of the applications such as text processing, image recognition, natural language processing and so on. Overall, pattern matching is one of the powerful tools for extracting information from

complex data sources, and also it is used in many of the applications of computer science and programming.

8) Complete this sentence : "NP problems are hard to solve but easy to _____"

Ans) NP problems are hard to solve but easy to ~~verify~~ check.

9) What is the example of an NP problem used in the video?

Ans) The example of Non-Polynomial (NP) problem used in the video is factoring.
Factoring is the concept of finding the prime factors of a number, which seems to be difficult for bigger integers. So, this means that NP problems are hard to do.

10) What are the TV shows mentioned in the video?

Ans) The TV shows which are mentioned in the video are 'The Simpsons' and 'Futurama'

11) Floating point numbers are essentially what?

Ans) Floating point numbers are essentially scientific notation

12) Computers perform scientific notation in what base?

Ans) Computers basically perform scientific notation in base 2. That's what the floating point is. This indicates that rather than the ten digits required in base-10 decimal notation, integers are expressed using just two digits, 0 and 1.
In binary scientific notation, the number is represented as a mantissa (or significand) multiplied by 2, which is raised to exponent.

13) What is the problem with adding 1/3 + 1/3 + 1/3 using base 10 and ignoring recurring numbers?

Ans) 1/3 + 1/3 + 1/3

$0.\overline{3} + 0.\overline{3} + 0.\overline{3} = 1$ (Not the exact sum)

Floating point math is essentially significant figure Computer will look at it and

0.3333333333 + 0.3333333333 + 0.3333333333

= 0.9999999999

But after a while, it will stop because it is running out of digits.

It is a floating point rounding error

Overall, what we can say is, when ignoring recurring numbers and adding the fractions in base 10 leads to inaccurate results, mainly while working with fractions which can't be represented using base 10.

14) What is 1/10 in base 2?

Ans) It is impossible to precisely represent 1/10 in base 2 with a finite amount of binary digits. This is due to the fact that

1/10 is a repeating decimal in base 10 and transforms into an infinitely repeating binary fraction when we try to convert it to base 2.

We use a method called "binary long division" or "division by 2" to convert 1/10 to binary.

Steps performed for "Binary long division" for 1/10:

$0.1 \times 2 = 0.2$ (integer part is 0)
$0.2 \times 2 = 0.4$ (integer part is 0)
$0.4 \times 2 = 0.8$ (integer part is 0)
$0.8 \times 2 = 1.6$ (integer part is 1)
$0.6 \times 2 = 1.2$ (integer part is 1)
$0.2 \times 2 = 0.4$ (integer part is 0)
$0.4 \times 2 = 0.8$ (integer part is 0)
$0.8 \times 2 = 1.6$ (integer part is 1)
$0.6 \times 2 = 1.2$ (integer part is 1)
$0.2 \times 2 = 0.4$ (integer part is 0)
.....

Here, this keeps on repeating the sequence '0011'.

Therefore the binary representation of 1/10 in base 2 is as follows:-

$0.0001100110011001100110011....$

15) What is the name of the function discussed in the video?

Ans) The name of the function discussed in the video is 'Recursive function'.

16) Can Ackermann's function be coded using for or 'DO' loops?

Ans) Yes, it is possible to code Ackermann's function using for or 'DO' loops, though this may need a lot of iterations and may not be effective for large inputs.
Ackermann's function is known as recursive function which computes a non-negative integer value from two non-negative integer parameters, typically represented as m and n.

17) What is the value of Ackermann(4,1)?

Ans) The Ackermann function is called as recursive function, which is defined for m and n (non-negative integers)

Ackermann(m,n) is recognized to be a computationally challenging function to evaluate for large values : m and n because its value increases very quickly as m and n increase.

The value of Ackermann(4,1) = 65533
This value is calculated with the help of recursive algorithm.

18) How many minutes will the machine in the video take to calculate Ackermann(4,2)?

Ans) It will take $2^{(65533)} \times 3$ minutes
$= 3 \times 2^{65,533}$ minutes
to calculate Ackermann(4,2) value
(It is a very big number)

19) The performance characteristic of Ackermann's function is described as what?

Ans) The performance characteristic of Ackermann's function is described as "hyper-exponential growth". This indicates that as the input values m and n rise, the function's value

increases very quickly.

In fact, the function's growth is happening very quickly that it outpaces the growth of other well-known functions like exponential or factorial functions. Because of this quick expansion, the Ackermann function is utilized in computability theory as a theoritical concept rather than for actual calculations.

20) A loop nested in another loop has the performance characteristic of what?

Ans) A loop nested in another loop has the performance characteristic of "quadratic time complexity" or O(n^2) time complexity.
When a loop is nested inside another loop, the inner loop will carry out its whole set of instructions for each time the outer loop iterates.
As a result, the number of iterations for the inner loop will rise in direct proportion to those for the outer loop.

21) What was the limitation of fortran mentioned in the video?

Ans) Fortran didn't do user-level recursion. In fortran, we might be expecting that every time it called itself, it would lay out a data area for each recursive call they are called "stack frames". You get lots of stack frames, one on top of another and as you come back through recursion, they are deleted and thrown away and you climb back to main program. FORTRAN doesn't

do that. It sets aside one stack frame. We should keep calling recursively it just tramples in its muddy gumboots over all the data area and we end up with total garbage. It no more gives the values of Ackermann function than fly to the moon.

22) What real world use needs complex recursion?

Ans) Not many things need it

23) There was a need to have a language that could cope with what?

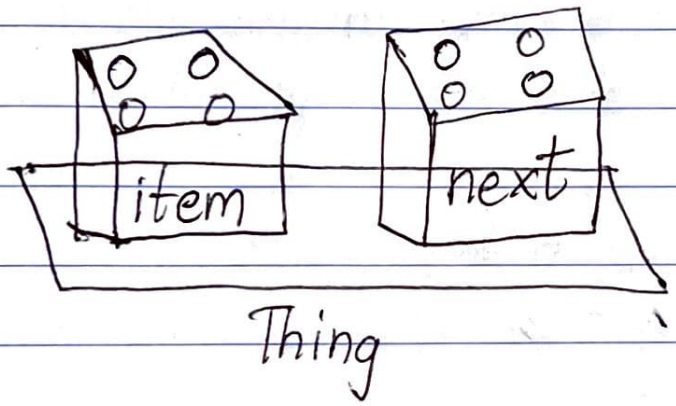
Ans) There was a need to have a language to evolve that could cope with different widths of objects.

24) C is most powerful when considered as the classical what?

Ans) C language finds its maximum power for even existing if we think of it as being considered as the classical system implementation language.

25) What are the names of the two fields of the 'THING' structure?

Ans) Red box on the top of thing holds a (pointer to) string of characters
→ it is known as item.
The other box on the thing is going to point off to the next thing along in this singly-linked list.



Thing

```
typedef struct _thing
{
  char *item;
  struct_thing *next;
} THING;
```

26) What is the advantage of the 'Triple Ref Technique'?

Ans) The Triple Ref Technique's main advantage is that enables a function to modify a

pointer value that is defined outside of the function's scope. This is helpful when you want to allocate memory for a data structure inside a function and deliver the pointer to that data structure to the caller. Additionally, it enables functions to modify several pointers simultaneously, which can simplify some algorithms and improve the readability of the code.

27) What is the procedure used in the video to compare the data structures?

Ans) - He has executed some tests to see which is faster : an array or linkedlist. Tested on different computers: atari, pi and imac. On Atari, Linked list was faster than the array. When ran on Raspberry pi, Array was about twice as fast as the Linked list. So, in this way running on different computers the data structures are compared.

28) Why is the reverse array faster on Atari?

Ans) It just uses slightly different instructions, their instructions on the six 8000 that allow you to do a decrement, testing with zero branch which isn't all in one instruction. So you can actually make the code slightly more compact and run slightly faster again. It is a small enough time.

29) What would be the goal of requiring people to be exposed to coding?

Ans) Exposing people to coding gives them that awareness and appreciation, broadly they allow people to do what's often termed computational thinking, they see the world computationally and they understand the ways in which computers might allow in the world whether the people can become programmers or we can anticipate them all having to program for themselves, I think it is unlikely but I think appreciation is really useful for them. It will give them appreciation & it will become part of their life.

30) List 3 or more of the different sort algorithms mentioned in the video

Ans) Sort algorithms which were mentioned in the video are bubble sort, Quick sort, Selection sort and cocktail sort.

31) What is the 'Decision Problem'?

Ans) Mathematicians posed this problem - in logic we are interested in finding "Do these premises entail this conclusion?" Premises → bits you start off with an argument
Conclusion → bits you want to establish the bit that you reason to with your argument.
We want to know "is there a test that will tell us yes for sure these premises do or don't entail this conclusion?"
"Is there an automatic way of finding out whether they

do or don't? This is known as "Decision Problem"

32) An example of an abstraction used in the video is, "A transistor is a type of ___"?

Ans) A transistor is a type of "Switch".

33) Which video was the most interesting or your favorite?

Ans) I like the video which explains about Programming BASIC and Sorting. It is interesting, he explains the sorting very clearly using visual representation. I liked that concept. So, it was my favorite video among all the videos posted.